ass 2

February 10, 2023

1 Speed of computation

Python, being an interpreted language, tends to be slower than compiled languages like C or Fortran. Some other languages like Java and Julia tend to use Just-in-Time compilation which can give speedups, but Python also has the problem of being dynamically typed, which eliminates the possibility of many optimizations.

The `timeit` library provides functions to estimate the time taken to run a piece of code. It can automatically run the code multiple times to get better average results, and can be used to identify bottlenecks in your program. However, it should be used with care as it is not a detailed function-call-level profiler.

It can either be `imported` as a module where you can then explicitly called `timeit.timeit(func)` to estimate time for a function, or you can use the *magic syntax* in Python notebooks as shown below.

import numpy as np x = np.random.rand(10000,1)

[ ]:
```
def sumarr(x):
    sum = 0
    for i in range(len(x)):
    # for i in x:
        sum += x[i]
    return sum
x= [10,20,30,40]
print(sumarr(x))
%timeit sumarr(x)
```

[ ]:
```
import numpy as np
def npsumarr(x):
    return np.sum(x)
print(npsumarr(x))
%timeit npsumarr(x)
```

2 Solving equations by Gaussian elimination

Once you have constructed two matrices A and B to represent the system of linear equations

$$Ax = b$$

you can then proceed to solve the equations using the process known as Gaussian elimination.

It is assumed you already know how the process works, but to refresh your memory, you could use the reference material at LibreTexts.

Basically it involves making the A matrix *triangular* and ultimately into the shape of an identity matrix.

[ ]:
```
# Input matrices - the set of equations - 2 variables x1 and x2
A = [ [2,3], [1,-1] ]
B = [6,1/2]
print(A)
print(B)
```

[ ]:
```
# Normalize row 1
norm = A[0][0]
for i in range(len(A[0])): A[0][i] /= norm
B[0] = B[0]/norm
print(A)
print(B)
```

[ ]:
```
# Eliminate row 2 - A[1] - need to check and ensure div-by-zero etc doesnt␣
↪happen
norm = A[1][0] / A[0][0]
for i in range(len(A[1])): A[1][i] = A[1][i] - A[0][i] * norm
B[1] = B[1] - B[0] * norm
print(A)
print(B)
```

[ ]:
```
# Normalize row 2 - B[1] will now contain the solution for x2
norm = A[1][1]
for i in range(len(A[1])): A[1][i] = A[1][i] / norm
B[1] = B[1] / norm
print(A)
print(B)
```

[ ]:
```
# Sub back and solve for B[0] <-> x1
# This can be seen as eliminating A[0][1]
norm = A[0][1] / A[0][0]
# note that len(A) will give number of rows
for i in range(len(A)):
    A[i][1] = A[i][1] - A[i][0] * norm
    B[i] = B[i] - A[i][0] * norm
print(A)
print(B)
```

2.1 Problems with Gaussian elimination

There are several obvious problems with the method outlined here. These include:

- Performance - Python lists are not the most efficient way to store matrices
- Zeros: the simple example does not consider a scenario where one of the values on the diagonal may be 0. Then some shuffling of rows is required.
- Numerical stability: there are several *normalization* steps involved, where it is quite possible for the values to blow up out of control if not managed properly. Usually some kind of pivoting techniques are used to get around these issues.

[ ]:
```
import numpy as np
A1 = np.array( [ [2,3], [1,-1] ] )
B1 = np.array( [6, 1/2] )
np.linalg.solve(A1, B1)
```

3 SPICE basics

Our goal is to implement a SPICE simulator. In order to do this, we first need to read in the circuit description from a text file. To start with, we will only consider the basic elements of SPICE: Voltage sources, Current sources, and Resistors. A typical SPICE netlist would look like this:

```
.circuit
R1 GND 1 1
R2 1 2 1
V1 GND 2 dc 2
.end
```

This is basically a *netlist* with 3 *nodes* - one of which is Ground (GND) which is assumed to be have a voltage of 0V. We can write down Kirchhoff's current law (KCL) equations at each node, to account for current balance. In addition, we will have some equations that specify the voltages between nodes having a direct voltage source, since there is no resistance there to provide an equation.

For the above example, the equations will be

$$\begin{aligned} \frac{V1-0}{R1} + \frac{V1-V2}{R2} &= 0 \\ \frac{V2-V1}{R2} + I1 &= 0 \\ V2 - 0 &= 2 \end{aligned}$$

which can be written in Matrix form as:

$$\begin{bmatrix} \frac{1}{R1}+\frac{1}{R2} & \frac{-1}{R2} & 0 \\ \frac{-1}{R2} & \frac{1}{R2} & 1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} V1 \\ V2 \\ I1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 2 \end{bmatrix}$$

At this point, you have reduced the solving of the SPICE equations to a known problem (linear equation solving) that you already know how to do.

3.1 AC sources

The assumption above is that the system consists only of Voltage or Current sources and resistors. What about capacitors, inductors, and AC sources? These can be handled in exactly the same way as long as the circuit is operating at a single frequency. We then replace the elements with their corresponding *impedance* values, which are frequency dependent complex numbers, but since there is only a single frequency they will still be constants.

3.2 Problem scenarios

- Voltage source loops
- Current sources at a node
- Circuit defined with both DC and AC sources
- Syntax errors

4 String and File manipulation

Given a SPICE netlist like the one above, you need to *read* it and construct an internal matrix as described above. For string manipulation, there are a few helpful utility functions that we can see here.

[ ]:
```
circ = """.circuit
R1 GND 1 1
R2 1 2 1
V1 GND 2 dc 2
.end
""".splitlines()

for l in circ:
    if l[0] == 'R':
        print("Found a resistor")
    elif l[0] == 'V':
        print("Found a voltage source with value: ", l.split()[4])
```

4.1 Files

You can read from a file using the `readlines()` method of file objects. One thing to keep in mind is how you open and close file objects. In particular, it is strongly recommended to use the pattern `with open("filename") as f:` to ensure that the file is closed once you are done with reading it.

[ ]:
```
import numpy as np
def factorial(N):
    if N<0: return "Factorial doesn't exist"
    elif N==0: return "1"
    else:
        mul = 1
        for i in range(1,N+1):
            mul=np.multiply(i,mul)
```

```
        return mul
print(factorial(5))
%timeit factorial(N)
```

5 Factorial

- The function calculates the factorial using a for loop, where mul is the running product and np.multiply is used to multiply i and mul at each iteration.
- The %timeit magic command is used to evaluate the execution time of the function.

[121]:
```
def uppertriangle(A,B):
    try:
        for i in range(len(A)):
            if i == N-1 and A[i][i] !=0:
                B[i] = B[i]/A[i][i]
                A[i][i] = A[i][i]/A[i][i]
            else:
                if A[i][i] == 0:
                    for p in range(i+1,len(A)):
                        if A[p][i] !=0:
                            A[i],A[p] = A[p],A[i]
                            B[i],B[p]=B[p],B[i]
                            break
                norm = A[i][i]
                B[i] = B[i]/norm
                for j in range(i+1,N):
                    norm1 = A[j][i]
                    B[j]=B[j]-(B[i])*norm1
                    for k in range(len(A[i])):
                        A[i][k] = A[i][k]/norm
                        A[j][k] = A[j][k] - (A[i][k]) * norm1
                    norm = A[i][i]
        return A,B;
    except ZeroDivisionError:
        print("The system of linear equations are inconsistent")
```

[122]:
```
def lowertriangle(A,B):
    try :
        uppertriangle(A,B)
        for a in range(len(A[0]),0,-1):
            for b in range(a-1,0,-1):
                norm2 = A[b-1][a-1]/A[a-1][a-1]
                A[b-1][a-1] = A[b-1][a-1] - norm2
                B[b-1] = B[b-1] - B[a-1]*norm2
        return A,B;
    except ZeroDivisionError:
        print("The system of linear equations are -inconsistent")
```

[123]:
```
def lnreq_sol(A,B):
    lowertriangle(A,B)
    return(A,B);
```

[ ]:
```
try:
    N=input("No of rows=")
    N = int(N)
    A = []
    B = []
    for i in range(0,N):
        A.append(input())
        A[i] = A[i].split()
        A[i] = [complex(k) for k in A[i]]
    B=input().split()
    B=[complex(k) for k in B]
    lnreq_sol(A,B)
    n=0
    for i in range(0,N):
        if len(A[i]) != len(A[0]):
            n+=1
            break
    if len(B) != len(A[0]) :
        print("The length of B matrix should be equal to length of each row in␣
 ↪matrix")
    else:
        if n==0:
            lnreq_sol(A,B)
            print(A)
            print("The solution to the system of linear equation is:")
            print(B)

        else:
            print("The length of each row should be same ")
except ValueError:
    print("The input should contain only string with numbers and spaces only")
%timeit
```

5.1 Problems with Gaussian elimination

There are several obvious problems with the method outlined here. These include:

- Performance - Python lists are not the most efficient way to store matrices
- Zeros: the simple example does not consider a scenario where one of the values on the diagonal may be 0. Then some shuffling of rows is required.
- Numerical stability: there are several *normalization* steps involved, where it is quite possible for the values to blow up out of control if not managed properly. Usually some kind of pivoting techniques are used to get around these issues.

```
[ ]: import math
FILENAME=input("Enter the filename:")
text_file = open(FILENAME, "r")
#read whole file to a string
data = text_file.read()

#close file
text_file.close()
data=data.splitlines()
print(data)
A = []
B = []
C = []
K =[]
n = 0
w=0
m=0
for j in range(len(data)):
    K.append(data[j])
    K[j]=K[j].split()
    if K[j][0] != ".circuit" and K[j][0] != ".end" and K[j][0] != ".ac" :
        if K[j][1] == "GND":
            if n<float(K[j][2]): n = float(K[j][2])
        elif K[j][2] == "GND":
            if n<float(K[j][1]) : n = float(K[j][1])
        elif K[j][1] != "GND" and K[j][2] != "GND":
            if n<max(float(K[j][1]),float(K[j][2])): n=␣
 ↪max(float(K[j][1]),float(K[j][2]))
    if K[j][0][0] == "V":
        m=m+1
    elif K[j][0] == ".ac":
        w = float(K[j][2])
n = int(n)
print(m)
print(n)
l = m+n
print(int(l))
print(f'The order of matrix is of {l} X {l}')
A=[[complex(0) for _ in range(int(l))]for _ in range(int(l))]
B=[complex(0) for _ in range(int(l))]
for i in range(len(data)):
    data[i]=data[i].split()
    print(data[i])
    if data[i][0][0] == "R" :
        if data[i][1] == "GND":
            A[int(data[i][2])-1][int(data[i][2])-1] += 1/ complex(data[i][3])
        elif data[i][2] == "GND":
```

```
            A[int(data[i][1])-1][int(data[i][1])-1] += 1/ complex(data[i][3])
        else:
            A[int(data[i][1])-1][int(data[i][1])-1] += 1/ complex(data[i][3])
            A[int(data[i][2])-1][int(data[i][2])-1] += 1/ complex(data[i][3])
            A[int(data[i][1])-1][int(data[i][2])-1] += -1/ complex(data[i][3])
            A[int(data[i][2])-1][int(data[i][1])-1] += -1/ complex(data[i][3])

    if data[i][0][0] == "C" :
        if data[i][1] == "GND":
            A[int(data[i][2])-1][int(data[i][2])-1] +=␣
↪(complex(data[i][3])*w)*1j
        elif data[i][2] == "GND":
                A[int(data[i][1])-1][int(data[i][1])-1] +=␣
↪(complex(data[i][3])*w)*1j
        else:
            A[int(data[i][1])-1][int(data[i][1])-1] +=␣
↪(complex(data[i][3])*w)*1j
            A[int(data[i][2])-1][int(data[i][2])-1] +=␣
↪(complex(data[i][3])*w)*1j
            A[int(data[i][1])-1][int(data[i][2])-1] +=␣
↪-(complex(data[i][3])*w)*1j
            A[int(data[i][2])-1][int(data[i][1])-1] +=␣
↪-(complex(data[i][3])*w)*1j

    if data[i][0][0] == "L" :
        if data[i][1] == "GND":
            A[int(data[i][2])-1][int(data[i][2])-1] += -1j/complex(data[i][3])*w
        elif data[i][2] == "GND":
            A[int(data[i][1])-1][int(data[i][1])-1] += -1j/complex(data[i][3])*w
        else:
            A[int(data[i][1])-1][int(data[i][1])-1] += -1j/complex(data[i][3])*w
            A[int(data[i][2])-1][int(data[i][2])-1] += -1j/complex(data[i][3])*w
            A[int(data[i][1])-1][int(data[i][2])-1] += 1j/complex(data[i][3])*w
            A[int(data[i][2])-1][int(data[i][1])-1] += 1j/complex(data[i][3])*w

    if data[i][0][0] == "I" :
        if data[i][1] == "GND":
            if data[i][3]== "dc":
                B[int(data[i][2])-1] +=  complex(data[i][4])
            else:
                B[int(data[i][2])-1] +=  complex(data[i][4])*complex(math.
↪cos(float(data[i][5])),math.sin(float(data[i][5])))
        elif data[i][2]== "GND":
            if data[i][3]== "dc":
                B[int(data[i][1])-1] += -1*complex(data[i][4])
            else:
```

```
                B[int(data[i][1])-1] +=  -1*complex(data[i][4])*complex(math.
↪cos(float(data[i][5])),math.sin(float(data[i][5])))
        else:
            if data[i][3]== "dc":
                B[int(data[i][1])-1] +=  -1*complex(data[i][4])
                B[int(data[i][2])-1] +=  complex(data[i][4])
            else:
                B[int(data[i][1])-1] +=  -1*complex(data[i][4])*complex(math.
↪cos(float(data[i][5])),math.sin(float(data[i][5])))
                B[int(data[i][2])-1] +=  complex(data[i][4])*complex(math.
↪cos(float(data[i][5])),math.sin(float(data[i][5])))

    if data[i][0][0] == "V" :
        n+=1
        if data[i][1] == "GND":
            A[n-1][int(data[i][2])-1] += 1
            A[int(data[i][2])-1][n-1] += 1
            if data[i][3]== "dc":
                B[n-1] +=  complex(data[i][4])
            else:
                B[n-1] +=  complex(data[i][4])*complex(math.
↪cos(float(data[i][5])),math.sin(float(data[i][5])))
        elif data[i][2] == "GND":
            A[n-1][int(data[i][1])-1] = -1
            A[int(data[i][1])-1][n-1] = -1
            if data[i][3]== "dc":
                B[n-1] +=  complex(data[i][4])
            else:
                B[n-1] +=  complex(data[i][4])*complex(math.
↪cos(float(data[i][5])),math.sin(float(data[i][5])))
        else:
            A[n-1][int(data[i][1])-1] = -1
            A[n-1][int(data[i][2])-1] = 1
            A[int(data[i][1])-1][n-1] = -1
            A[int(data[i][2])-1][n-1] = 1
            if data[i][3]== "dc":
                B[n-1] +=  complex(data[i][4])
            else:
                B[n-1] +=  complex(data[i][4])*complex(math.
↪cos(float(data[i][5])),math.sin(float(data[i][5])))

N = l
lnreq_sol(A,B)
print(A)
print("solution Matrix of MNA is:")
print(B)
```

6 netlist solver

- We took the imformation of circuit by giving the file directory or name.
- first for loop calculates the no of voltage sources and no of node which indirect gives size of matrix
- In second circuit we go by line by line arrange the element value to their respective places in matrix A and independent sources are placed in respective positions in matrix B.
- We call matrix solver at end to solve the A and B by function `lnreq_sol(A,B)` by giving A,B and N(size of matrix)
- for capacitor and inductor it takes admitance(jwc and 1/jwl respectively). -For .ac line it takes the frequency of circuit for circuit having Ac components.

7 Linear equation solver

- This programme attempts to use lower triangular form to solve the system of linear equations represented by square matrix A and vector B. The matrix is first converted to upper triangular form by calling the uppertriangle function, and then it is converted to lower triangular form by calling the lowertriangle function.
- The matrix A and vector B are first entered into the code, followed by the number of rows N. There should only be integers and spaces in the input. The input matrix A is stored as a 2D list of complex numbers, and the vector B is stored as a list of complex numbers.
- At first i took first row of matrix A and and performs the Gaussian elimination process. to get the fimal solution i took the last row of gaussian matrix and perform the colume operation with the diagonal element. after all colume operation I took last second row diagonal element and porforming the same operation. we end the process till we get first row diagonal element and the produced matrix is solution matrix.

8 Assignment

The following are the problems you need to solve for this assignment. You need to submit your code (either as standalone Python script or a Python notebook), a PDF document explaining your solution (either generated from the notebook or a separate LaTeX document), and any supporting files you may have (such as circuit netlists you used for testing your code).

- Write a function to find the factorial of N (N being an input) and find the time taken to compute it. This will obviously depend on where you run the code and which approach you use to implement the factorial. Explain your observations briefly.
- Write a linear equation solver that will take in matrices A and b as inputs, and return the vector x that solves the equation $Ax = b$. Your function should catch errors in the inputs and return suitable error messages for different possible problems.
  - Time your solver to solve a random 10×10 system of equations. Compare the time taken against the `numpy.linalg.solve` function for the same inputs.
- Given a circuit netlist in the form described above, read it in from a file, construct the appropriate matrices, and use the solver you have written above to obtain the voltages and currents in the circuit. If you find AC circuits hard to handle, first do this for pure DC circuits, but you should be able to handle both voltage and current sources.

8.1 Bonus assignments

- (Small bonus): after reading in the netlist, allow some or all sources and impedances to be controlled interactively - either using widgets or other mechanisms. On each change you should recompute the currents and voltages and display them.
- (Large bonus): make a solver that can do real-time transient simulations of a SPICE netlist and update the currents and voltages dynamically. They should also be plotted as a function of time and react to changes. This is something along the lines of https://www.falstad.com/circuit/. Ideally you should be able to do a real-time demo of some experiments you might conduct as part of a basic electronics lab, and simulate the behaviour of an oscilloscope and signal generator.